Interactive out-of-core visualization of very large multiresolution time series scientific data
Progress Report
April 10, 2008

1. Goals for year 2

The principal goals for the year as described in the year 1 progress report are shown below.

Development Task	Month	Science task
1. Evaluate VAPOR	3	Give us advice/feedback on value of *VAPOR* visualization tools
2. Expand non-uniform grid support	5	None. Their data format is already supported.
3. Package *STARgen* and *VisIt* interface for distribution	6	Test ease of installation.
4. *VisIt*: auto resolution change	6	Use and evaluate modified interface
5. Error support	6	Use and evaluate error functionality
6. Granite/C++: caching/prefetching	10	Use caching/prefetching features, compare results
7. Adaptive resolution prototype	10	Test our implementation tool and give feedback

2. Achievements for year 2

2.1. Evaluate VAPOR

We downloaded, installed, and experimented with *VAPOR* (**V**isualization and **A**nalysis **P**latform for **O**cean, Atmosphere and Solar **R**esearchers (Clyne and Rast, 2007). This package supports multiresolution data representation for large scale fluid flow visualization. In many ways the strong features of this software are complementary to those of *VisIt*. The focus on three-dimensional flow visualization provides substantially better support in this area (and more convenient support) than is available in *VisIt*. For this reason, we thought that it might be a potential alternative to or supplement for *VisIt.* After a thorough analysis of the functionality and extensibility of the software, we decided that it would not be productive to put additional effort into *VAPOR* at this time. The key factors for this decision included:

1. A major goal of using *VAPOR* was the hope that we could easily embed *our* multi- and adaptive resolution data model seamlessly into the system. If this were possible, we believed we could broaden the multiresolution tools available to our science colleagues with modest effort. Our evaluation of the source code indicated that the integration effort was going to be quite a bit more complicated than we had hoped (dreamed?).
2. Our science colleagues are not interested in using *VAPOR*. They are committed to the *VisIt* environment primarily because of the support it provides for parallel computation on a workstation farm. This functionality is not currently available in *VAPOR.*

Clyne, J. and Rast, M. "A prototype discovery environment for analyzing and visualizing terascale turbulent fluid flow simulations", in proceedings of **Visualization and Data Analysis 2005**, pp. 284-294, January 2005.

2.2. Non-uniformly distributed grid

We implemented support in our *VisIt* interface for unstructured grids (non-uniformly distributed data points).

2.3. Package STARgen for the *VisIt* multiresolution data interface

2.3.1. VisIt database plugin for multiresolution data

We have implemented a database plugin for *VisIt* that can read our multiresolution hierarchy and provide *VisIt* renderers with multiple resolutions of data. With this interface any *VisIt* rendering pipeline can access our multiresolution data format. The user explicitly controls the resolution level with a simple interactive control panel.

2.3.2. STARgen multiresolution data generation utility

We also have a general purpose command-line utility, called *STARgen*, that allows a scientist to generate arbitrarily complex hierarchies of spatial and temporal resolutions and to define how the many output files are organized. This flexibility is intended to allow scientists to be able to match the output data organization to the conventions of the environment in which the files will be used. This tool is used to generate the data formats that are supported by our *VisIt* multiresolution database plugin. The database plugin, however, only supports a small subset of the possible data organizations that *STARgen* can generate. Since we are generating the data for the plugin, there is no reason to require the plugin to support multiple file organizations – one will do.

2.3.3. STARgui – a user-friendly interface to STARgen

In order to simplify the scientist's access to the data generation tools, we implemented a GUI environment to serve as an interface to the not-so-friendly command-line oriented *STARgen* tool. This tool guides a user through the process needed to create the appropriate metadata information required for *STARgen* and then through the process of specifying the desired operations. A screen shot of one stage from *STARgui* is shown in figure 1.

2.3.4. Distributable software

The *STARgui / VisIt Database Plugin* combination represents a useful standalone utility that could be of value to a wide range of scientists. We have packaged up an initial version of these tools with documentation into a form that can be easily downloaded from our web site. The *STARgui* tool is an executable Java *jar* file; the *STARgen* functionality is implemented by very basic C++ code that makes use of no non-standard libraries. Both of these components should be easily portable; we have installed them on both Linux and MacOS systems. The *VisIt* plugin, of course, requires the same elaborate environment as *VisIt*.

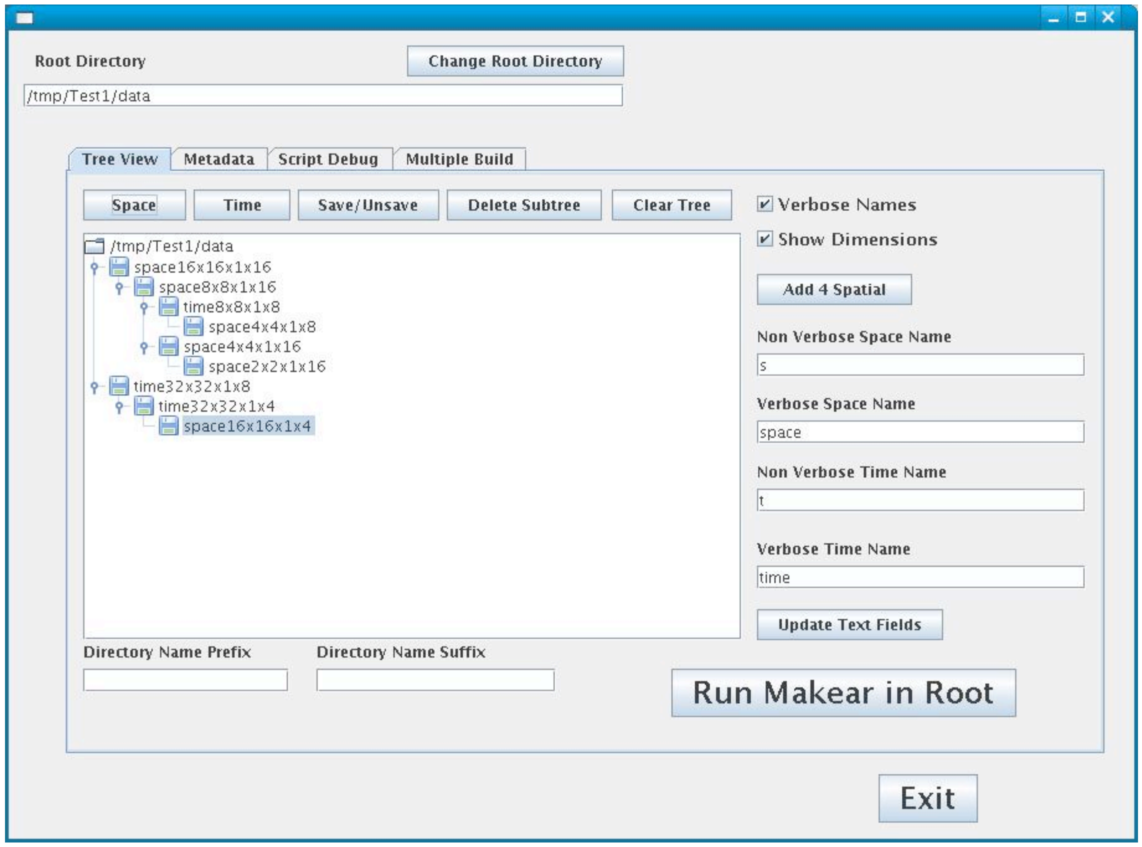
The software can be downloaded from http://www.cs.unh.edu/star/download.htm. There are actually three different tools available that can be used together or independently:

STARgui – a Java based gui that allows the user to build hierarchies of arbitrary mixtures of spatial and temporal resolutions.

STARvisit – a *VisIt* plugin for *STAR* data hierarchies

STAROpenCCGM – a special purpose utility to generate STAR data from the OpenCCGM simulator output data

The currently available versions limit the data to 3 resolution levels because of the current *VisIt* plugin. We plan to remove this restriction in the next version.



Root Directory: /tmp/Test1/data

```
/tmp/Test1/data
  space16x16x1x16
    space8x8x1x16
      time8x8x1x8
        space4x4x1x8
      space4x4x1x16
        space2x2x1x16
  time32x32x1x8
    time32x32x1x4
      space16x16x1x4
```

☑ Verbose Names
☑ Show Dimensions

Non Verbose Space Name: s
Verbose Space Name: space
Non Verbose Time Name: t
Verbose Time Name: time
Directory Name Prefix:
Directory Name Suffix:

Figure 1. A screen shot from *STARgui.*

2.4. VisIt autoresolution change

One goal for this year that we had thought would be straightforward was to provide an automatic resolution change as a user zooms in or out. This would be a convenient mechanism to make the transition between resolution levels implicit and seamless (given adequate performance). We have implemented this feature outside of *VisIt* and it appears to be an effective model for interaction with multiresolution data. Unfortunately, implementing this feature at the application level does not seem to be possible within the *VisIt* framework. There is currently no mechanism that allows a database plugin to be contacted with information about the state of the viewing parameters. A fundamental assumption within the *VisIt* kernel is that a database plugin loads all the data needed for the visualization before any rendering takes place and no further interaction occurs with the plugin.

The only solution available to us is to modify the low level *VisIt* code to install the "hook" we need; we do not think that this is a viable long-term option because of significant maintenance and portability issues. We still plan to implement such a hook to evaluate the usability of this feature in *VisIt*, but we do not plan to make it available outside our environment. If our changes are minor and the functionality is effective, we will request that the *VisIt* project incorporate such a feature in future releases.

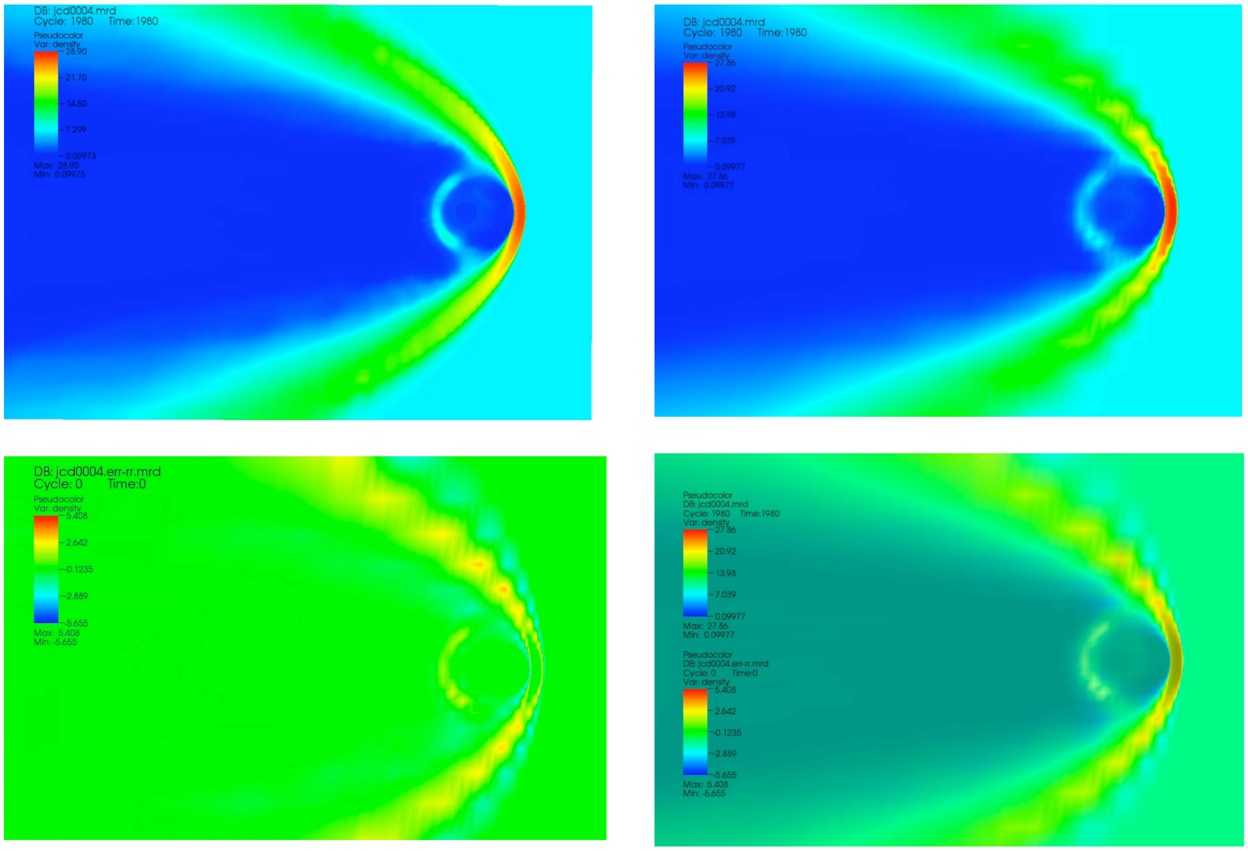


Figure 2. One frame of an MHD simulation using our *VisIt* database plugin and error data. The upper left shows the original data; the upper right image shows the data at one lower resolution level; the lower right shows the error associated with the lower resolution data; and the lower right shows the error superimposed over the lower resolution image with an opacity of about 60%.

2.5. Error representation

2.5.1. Error generation and visualization tools

We have implemented a prototype version of software to support our error model for multiresolution data. This tool generates accurate error information for each resolution of our data hierarchy when the hierarchy is generated using wavelet transformations. The error data has the same resolution as the data at that resolution and represents the approximate *local* error that occurs if this resolution is used to reconstruct the original data. We can generate both *maximum* and *average* error information; each are valuable for different kinds of user goals. The error is of the same format as the data and can be rendered by any *VisIt* rendering tool. Figure 2 shows 4 images from a frame produce by an MHD simulation using our multiresolution database plugin and rendered using standard *VisIt* modules. The upper left image shows the original high-resolution data for that frame; the upper right image shows the data at one resolution coarser than the original data; the lower right image shows the average local error associated with the lower resolution data; and the lower right image shows the error data displayed as a semi-transparent image on top of the low resolution data. The superimposed image is most informative while interactively changing the opacity, and it is also effective to just flip back and forth between the error image and the low resolution image.

Our wavelet-based multiresolution data generation utility produces data points at *new* positions in space that do not correspond to any of the original data positions. This characteristic complicates the implementation of adaptive resolution data sets. Consequently, it is also desirable to be able to create multiresolution data hierarchies using *decimation* strategies in which each lower resolution representation is defined by a subset of the data locations in the next higher resolution representation. We have implemented such a tool and will soon complete the corresponding error generation utility.

2.5.2. Expanded error model

Although the existing prototype for error representation is an important accomplishment, we believe we now have a "better" idea. We now realize that maintaining error at the same resolution as the data is often overkill. Especially, for very large datasets, the important role of the error information is to provide feedback that tells the scientist when to use a higher resolution representation. Seldom is the scientist interested in the error at a specific *point* in the dataset; instead, we need to provide a variety of error representations for *regions* of the data.

We have developed a new error model that incorporates many different error representations, but at lower resolution than the data resolution to which it is applied. For example, we plan to maintain several error measures, such as maximum error, average error, standard deviation, and signal-to-noise ratio information, but this information will be at 2-4 resolutions coarser than the data to which it applies. Summary measurements, such as SNR and standard deviation, are very important tools for understanding the error distribution, but these need multiple data points to produce meaningful values. We still want to provide this information "locally" since it is the local error that determines whether the scientist will zoom in to the higher resolution *in that region.* An error resolution that is just 2 steps coarser than the data to which it applies can be represented using 1/64 of the memory needed to represent error at the same resolution as the data. This is a significant savings for the first few resolution levels for very large data sets, *and* there are significant advantages to being able to present aggregate (but still local) error information to the scientist.

We have begun the implementation of software to support this expanded error model.

2.6. Granite/VisIt interface

Granite is our experimental scientific database package that provides extensive support for efficient access to large data sets in a Java-based environment. This system particularly is particularly motivated by the goal to support interactive *out-of-core* data visualization. We have developed several techniques to support this goal including the MR and AR data models and caching/pre-fetching techniques for multi-dimensional data. All of these techniques aim to speed up I/O, the primary performance bottleneck to achieving our goals. Performance improvements come by pre-fetching necessary data once and retaining it until it is no longer needed to avoid re-reads, and by reducing the volume of data accessed, as multi and adaptive resolution do.

2.6.1. Caching/prefetching

Our goal to interface our Granite caching/prefetching code to the *VisIt* environment continues to be unmet. The rigid complexities of the *VisIt* data model (at the present time) preclude any effective *out-of-core* rendering. As with the automatic resolution change, we are now planning to explore the difficulty of inserting hooks into the underlying *VisIt* control software to allow the datbase plugin to update the internal data as the rendering takes place. If we are successful, we may not want to support this feature, but we will encourage that these changes be incorporated into future *VisIt* releases.

2.6.2. Data compression

Although we are unable to access the Granite caching/prefetching facilities from *VisIt,* we have begun development of additional Granite functionality that will be directly usable via the *VisIt* data plugin that we have developed – block-oriented compression of data files. Data compression has long been used as a technique for reducing storage and I/O demands, but this usually just means encoding and re-inflating entire datasets.

Our approach is motivated by the fact that we want to support rapid access to specific regions (subblocks) of a large dataset without reading the entire file. In this context, it does not help to compress the entire file since this would mean reading everything and discarding what we don't need. Instead, we divide a large dataset into local partitions that we compress individually. With proper indexing, the appropriate partitions can be quickly identified, fetched and decompressed to satisfy a subblock query.

We want to incorporate compression seamlessly into our framework so the user is unaware of its behind-the-scenes role. This seamless integration should be orthogonal to other system components. We want the MR and AR data model implementations for both data and error representation to take advantage of compression where appropriate in a way that is transparent to other parts of the system.

There is a natural tradeoff between achieving high compression ratios (which favors large partitions), and reading and decompressing only what is necessary for the query (which favors small partitions). Preliminary experiments show that there is opportunity for a balanced choice for block size that achieves effective compression without reading too much extra data. Over the next few months we will conduct extensive tests our colleague's MHD data to develop guidelines to help them choose effective partition sizes given specific dataset sizes and distributions of query region size.

This functionality will appear at a lower level of our multi-layered architecture than the data model level. Thus we believe we can implement it in a way that supports our MR and AR data models transparently. In addition, the use of this functionality is consistent with the *VisIt* data model and we should be able to access it from our *VisIt* database plugin.

2.7. Adaptive resolution data

2.7.1. AR prototype

We made good progress on the design and planning of our AR software support prototype system. Implementation was about to begin when the principal person responsible for this task had to withdraw from the project and school for medical reasons. This has been a significant setback for this effort. We are still not back to where we thought we were in November, 2007.

2.7.2. Expanded AR data model

In parallel with the implementation design, we explored alternative data representation models that could provide more accurate data modeling in less space. This effort was motivated by our experience in evaluating and using our error tools in the context of our multiresolution data generation based on *wavelet* transformations.

Wavelet transforms generate two output files, the *summary* data and the *detail* data. Mathematically, the *detail* coefficients encapsulate the error resulting from using the *summary data* as an approximation to the original data. If all summary and detail coefficients are kept, the original data can be reconstructed in a *lossless* manner (except for some computational roundoff error). Keeping all the coefficients, however, does not result in any memory savings. Since reduced memory use (and especially reduced I/O cost) is the critical goal of our data model, we have been simply discarding all *detail* coefficients (7/8 of all coefficients with 3D data). This has little effect in regions of low variation in data magnitude, which is where detail coefficients are relatively small. In regions of high change, detail coefficients are larger and more error results from using only summary coefficients to represent the original data.

Our *adaptive resolution* data model is based on identifying regions of high error at a particular resolution and representing those regions by the wavelet *summary* coefficients at a higher resolution. We still delete the detail coefficients.

We have been experimenting with alternative approaches that save the detail coefficients at a lower *precision* than the summary data. We realized that most of the error in a low resolution data representation is caused by a relatively small number of very high detail coefficients. We also realized that the precision of the detail coefficients is not nearly as important as their magnitude in determining the error that it represents. In most cases, we believe that a *byte* or a *short* will be adequate for representing the error data. If the simulation output data is stored as a *float* and if the error is stored as a *short* at two resolution levels lower than the data, the memory cost for the error data will be significantly less than 1% of the memory cost for the data. If we use *byte*, the cost is less than 0.4%. Because the magnitude of the detail is far more important than the precision, we are planning to map the detail coefficient by a nonlinear function such as the ones shown in Figure 3. The best choice depends on the magnitude of the largest detail coefficient. This would be one additional piece of metadata needed for the decompression.

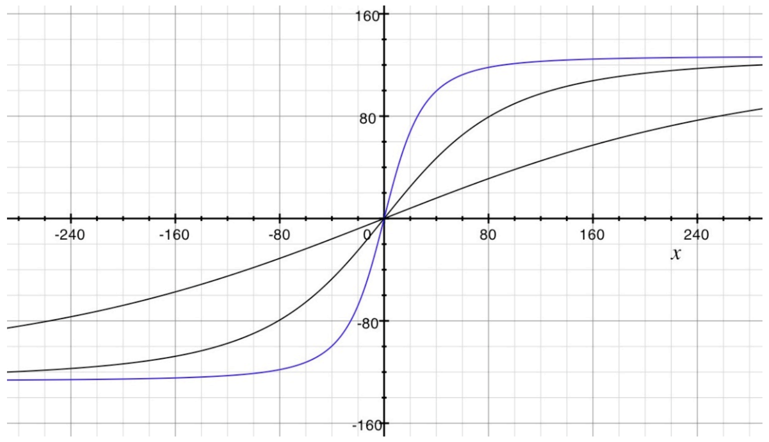


Figure 3. Possible mappings of a detail coefficient to a byte

2.8. Summary of our VisIt interface

Our *VisIt* interface is an effective mechanism for accessing multiresolution data sets without having to rewrite each visualization tool – as long as the data to be rendered fits in main memory. This is not a huge restriction given the basic interaction model whereby higher resolution data representations are accessed as the user zooms in to smaller spatial and temporal ranges – the system can use available memory as a constraint in determining which resolution to access for a given spatio-temporal request. However, there are significant features of our proposed environment that are not effective in this context. In particular, we have shown dramatic improvements in interactive visualization performance using our *iteration-aware* caching and pre-fetching techniques for *out-of-core* rendering. We cannot provide this functionality within the current *VisIt* environment.

2.9. Achievements summary

The table below provides a very terse summary of what we have accomplished for each of our major goals.

Revised Goal	Achievements
1. Evaluate VAPOR	Done. VAPOR's internal organization does not appear to be easily adaptable to our needs; our science colleagues are not particularly interested in the VAPOR functionality.
2. Expand non-uniform grid support	Completed for non-uniform rectilinear data distribution.
3. Package *STARgen* and *VisIt* interface for distribution	Completed the interface and created a GUI front end that makes the data generation much more user friendly
4. *VisIt*: auto resolution change	Not implemented; not feasible with current *VisIt* design.
5. Error support	A preliminary version of error representation software has been completed and is usable for evaluation purposes. We have made significant progress in developing a new, more workable and useful error model. Preliminary implementation has begun and we have had some encouraging results.
6. Granite/Visit interface: caching/prefetching	It does not appear that it is feasible to access Granite prefetching from *VisIt*. We have, however, made progress in developing a file block compression facility in Granite. This will be available to our *VisIt* database plugin and should improve I/O performance significantly.
7. Adaptive resolution prototype	Significant design and planning was completed. The implementation was interrupted because of a personnel issue. We have a replacement for that individual who will be responsible for this effort as of the summer.

3. Revised future goals

3.1. Extend VisIt/STARgen multiresolution data interface

Our initial *Visit*/*STARgen* utility supports three spatial resolution levels, each with its own temporal resolution. We will significantly relax these restrictions to allow the user to have arbitrary combinations of spatial and temporal resolution. Target completion: June 2008.

3.2. VisIt autoresolution change

We intend to implement the autoresolution change feature in *VisIt.* Although we do not plan to distribute this functionality, we will get feedback from our colleagues about its value and, if justified, lobby with the *VisIt* community to incorporate access mechanisms for functionality such as this into the core *VisIt* software. Target completion: June 2008.

3.3. Multiresolution data generation by decimation

We are extending our multiresolution data generation functionality to include a decimation option along with the current wavelet option. Even though the error introduced by decimation is normally higher than that introduced by wavelet decompositions, it is significantly easier and more efficient to implement adaptive resolution support for multiresolution hierarchies generated by decimation techniques. We will extend our *VisIt* database plugin (or implement another) to support this variation of multiresolution data. Target completion: September 2008.

3.4. Granite/VisIt interface

We plan to complete a preliminary version of Granite support for compressing large data files by blocks and incorporate this functionality into our *VisIt* database plugin. Target completion: September 2008.

3.5. Adaptive resolution support

Adaptive resolution (AR) data representation provides a single data representation that has different resolutions in different spatial or temporal regions; regions with high variation will use a higher resolution and than those with low variation. This model should allow for significantly reduced memory utilization and I/O time. Target completion: September 2008.

3.6. VisIt AR interface

Once we have a reasonable AR implementation, we want to provide access to it from the *VisIt* environment. Target completion: December 2008.

3.7. Error model implementation

During this past year, we have significantly modified and extended our error model for MR and AR data. Our major innovation is the realization that we need to provide a variety of error representations that rely on coarser resolutions of summary error characteristics. We will implement several statistical models of *local* error including mean, standard deviation, and signal-to-noise ratios. These will all be supported at multiple resolutions for each data resolution. Target completion: January 2009.

3.8. Final Evaluation

Although we have incorporated evaluation as a component of each of our proposed tasks, we plan to devote the last quarter of the grant period to a systematic effort to quantify the various aspects of our implementation. We are particularly interested in understanding the following:

1. how the various options for generating AR data representations affect the interactive data access performance for the particular MHD simulation datasets used by our colleagues;
2. what error statistics are effective and efficient for providing insight to the scientists;
3. although we are planning to rely on existing *VisIt* visualization tools for presenting the error, we know that effective visualization is a key component for providing insight; consequently, we will work with our science colleagues to identify visualization tools that help make the error representations more effective;
4. how effective the Granite block compression functionality is for the MHD simulation data and what block size guidelines seem to be most effective.

Target completion: March 2009.

3.9. Summary of major goals

The table below provides a summary of the major planned tasks for the remainder of the grant period. Note that there will be additional code packaging and distribution tasks for subsequent versions of the software. Specific dates for completion of these tasks will be determined as the development proceeds.

Development Task	*Date*	*Science task*
1. Extend *VisIt/STARgen* package	6/08	Give us advice/feedback on ease of installation/use.
2. Implement *VisIt* hook for auto resolution change	6/08	Use and evaluate resolution change interface
3. Investigate *VisIt* hook for out-of-core rendering	6/08	If successful, test performance.
4. MR data generation by decimation	9/08	Compare MR by decimation vs MR by wavelet.
5. Granite/*VisIt* interface for accessing compressed files.	9/08	Help evaluate performance characteristics and develop partition size heuristics for their data.
6. Adaptive resolution prototype	9/08	Advise on relative importance of temporal v. spatial resolution changes and appropriate error criteria for local resolution decisions
7. *VisIt* data interface: AR support	12/08	Compare performance/quality to non-AR using *VisIt*
8. Error model design/implementation	1/09	Advice on useful error representations and resolutions.
9. Final evaluation	3/09	Significant collaboration in the process of evaluation and the interpretation of the results.

We point out that there is one major previous goal that has been omitted from this list: an attempt to incorporate our software into an existing grid environment. Although this is still an overall goal of our research effort, we do not think that it is realistic to achieve this in the coming year. The loss of one of our key team members last fall has had a major negative effect on our accomplishments this year. Although we are beginning to recover, we simply will not be able to achieve all that we had originally intended.

4. Publications

Bergeron, R.D. and R.A. Foulks, "Interactive Out-of-Core Visualization of Multiresolution Time Series Data", in *Numerical Modeling of Space Flows*: *1ST IGPP – CalSpace International Conference*, ed. Nikolai V. Pogorelov and Gary P. Zank, ASP Conference Series, Vol 359, Astronomical Society of the Pacific, 2006, pp. 285-294.

Foulks, R., D. Benedetto, R.D. Bergeron and T.M. Sparr, STARdata: A Data Server for Multiresolution Time Series Data, *AGU Fall Meeting Abstracts,* Dec. 2006, p. A1316+.

Foulks, R.A. and R.D. Bergeron, Multiresolution Data Access Within The VisIt Visualization Environment, Proceedings of the NASA Science Technology Conference 2007, June 19-21, University of Maryland University College. http://esto.nasa.gov/conferences/nstc2007/papers/Foulks_Andrew_A11P1_NSTC-07-0066.pdf

STARgui User's Guide, http://www.cs.unh.edu/star/release/UsersGuide.doc.

Foulks, R.A. and R.D. Bergeron, Adaptive Resolution Data Access For Visualization of Magnetohydrodynamic Simulation Data, in preparation, http://www.cs.unh.edu/star/papers/inprogress.pdf .